MODULE 1: FOUNDATIONS OF THE WEB

Chapter 1: Introduction to Dynamic Web Content

1.1 The Technical Preface: The Evolution of Digital Communication

To understand the modern web, we must first define what it is *not*. In the early 1990s, the World Wide Web was essentially a global filing cabinet of digital brochures. These were **Static Web Pages**. If a company wanted to change a price on their website, a human being had to open the original code file, type the new price, save it, and manually upload it to a server.

The "Problem" was scale. As the internet grew, manual updates became impossible. The "Solution" was the birth of **Dynamic Web Content**. A dynamic website is not a fixed file; it is a program that "builds" a page on the fly every time someone requests it. When you log into an online bank, the page you see does not exist until the second you click "Login." The server fetches your specific balance from a database and "draws" the HTML page specifically for your eyes.

1.2 The Client-Server Model: A First Principles View

In IT, we talk about the "Client" and the "Server." This is the foundational architecture of the internet.

- **The Client:** This is usually your web browser (Chrome, Firefox, Safari). It is the "Requester." It sits on your device and waits for you to tell it where to go.
- **The Server:** This is a high-powered computer located in a data center. Its only job is to "Serve" or provide files when asked. It runs software like **Apache** or **Nginx** that listens 24/7 for incoming requests.

What happens when you type a URL?

Imagine you type http://www.google.com into your browser. Here is the microscopic journey of that action:

1. **The DNS Lookup:** Your computer doesn't know where "https://www.google.com/search?q=google.com" is. It only understands numbers (IP Addresses). It contacts a **Domain Name System (DNS)** server—essentially a global phonebook—and asks: "What is the IP address for https://www.google.com/search?q=google.com?" The DNS replies with something like 172.217.7.164.
2. **The TCP Handshake:** Your computer reaches out to that IP address. It performs a "Three-Way Handshake" (SYN, SYN-ACK, ACK). Think of this as two people shaking hands and agreeing: "I am going to send you data now," and "I am ready to receive it."

3. **The HTTP Request:** Your browser sends a formal letter called an **HTTP Request**. It says: "Dear Server, please give me the file located at /index.php. I am using a Windows computer and I prefer the English language."
4. **The Server Processing:** The server receives the letter. It sees the .php extension and realizes: "Wait, this isn't a static file. I need to run this code through the **PHP Engine**."
5. **The HTTP Response:** Once the code is finished running, the server sends back an **HTTP Response**. This contains a **Status Code** (like 200 OK) and the actual content of the page.

1.3 Deep Dive: The HTTP Protocol & Status Codes

HTTP (HyperText Transfer Protocol) is the language used by the Client and Server to talk. It is a "Stateless" protocol, meaning the server has no memory. Every time you click a link, the server treats you like a total stranger unless we use advanced techniques like "Sessions" (which we cover in Chapter 12).

Understanding Status Codes

The server always starts its response with a three-digit number. This is the "Status Code."

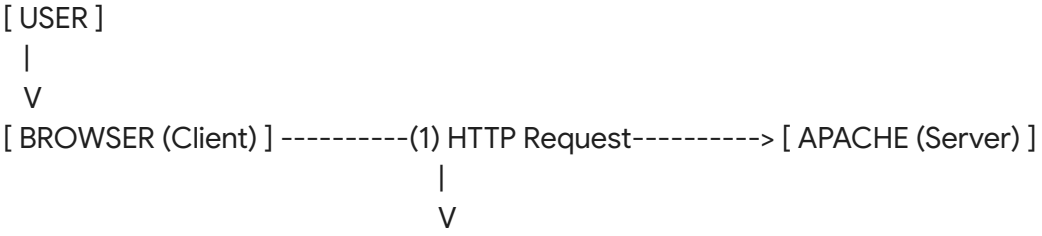
- **1xx (Informational):** "I've received your request, hold on a second."
- **2xx (Success):** "I found what you wanted, here it is!" (The most common is **200 OK**).
- **3xx (Redirection):** "The page has moved. I'm sending you to the new location."
- **4xx (Client Error):** "You did something wrong."
  - **404 Not Found:** You typed the URL wrong or the page was deleted.
  - **403 Forbidden:** You aren't allowed to see this folder.
- **5xx (Server Error):** "I (the server) crashed. My code has a bug."

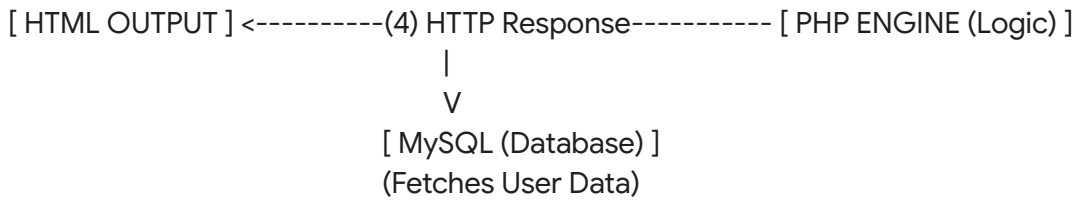
1.4 The Architecture of the "Big Three"

To build a full-stack system, you need three distinct layers working in harmony. This is often called the **Stack**.

1. **The Interface (HTML/CSS/JS):** This is the "Front-End." It is the only part the user ever sees. It is like the steering wheel and dashboard of a car.
2. **The Brain (PHP):** This is the "Back-End." It handles the logic. It checks if your password is correct, calculates your shopping cart total, and talks to the database.
3. **The Memory (MySQL):** This is the "Database." It stores all the permanent information, like your username, your past orders, and your profile picture URL.

1.5 Conceptual Diagram: The Data Flow





1.6 Practical Mini-Lab: Exploring the "Invisible" Web

You can see this process happening right now in your browser:

1. Open any website (e.g., wikipedia.org).
2. Right-click anywhere on the page and select **"Inspect"** or press F12.
3. Click on the **"Network"** tab.
4. Refresh the page (Ctrl + R).
5. You will see a list of every file the server sent you. Click on the first one (the domain name). Look for **"Response Headers."** You will see the Server type (e.g., "Apache") and the Status Code (e.g., "200").

1.7 Common Beginner Pitfalls

- **The "Double Click" Mistake:** Beginners often try to run a .php file by double-clicking it on their desktop. This will not work! PHP files must be "Served" by a web server. If you double-click it, your browser will just show you the raw code, not the finished website.
- **Case Sensitivity:** On many servers, Index.php and index.php are different files. Always use lowercase for your filenames to avoid "404 Not Found" errors.

1.8 Chapter 1 Review Questions

1. Define the difference between a "Static" and a "Dynamic" web page.
2. What is the specific role of the "DNS" in the web request cycle?
3. List the "Big Three" technologies of the web stack and explain their individual purposes.
4. If you receive a "403 Forbidden" error, what does that imply about the server's permissions?
5. Why is HTTP called a "Stateless" protocol?
6. What does the "SYN-ACK" part of a TCP handshake represent in plain English?
7. Which status code range indicates a problem with the server's code?
8. Explain the analogy of the "Client-Server" model using a restaurant as a reference.
9. What is the purpose of the httpd.conf file in the Apache web server?
10. Why can't a browser execute PHP code directly?

MODULE 1: FOUNDATIONS OF THE WEB

Chapter 2: Setting Up a Development Server

2.1 The Concept of a "Development Stack"

In professional software engineering, we never write code directly on a live website (the "Production" server). If you make a typo on a live site, the entire business goes offline. Instead, we build a Development Environment on our local computer.

To do this, we use a Stack—a collection of software working together. The industry standard is the LAMP Stack:

- **Linux (The Operating System)**
- **Apache (The Web Server software)**
- **MySQL (The Database engine)**
- **PHP (The Programming language)**

For students on Windows or macOS, we use "Bundles" like AMPPS or XAMPP which package these Linux-style tools into a single installer.

2.2 The Loopback Interface: Understanding `localhost`

When you set up a server on your own computer, you access it via the address `http://localhost` or the IP `127.0.0.1`.

The Technical Mechanic: This is called the Loopback Address. Normally, when a computer tries to connect to an IP, the data travels out through the network card to the internet. However, the IP `127.0.0.1` is hard-coded into every operating system to mean "Me." It tells the network card to turn the data around and send it right back into the machine's own internal software.

2.3 Visual Installation Guide: AMPPS

For this handbook, we recommend AMPPS because it closely mimics a professional production environment.

Step 1: Download and Installation

1. **Visit `www.ampps.com` and download the version for your OS.**
2. **Crucial Step (Windows): During installation, if your Firewall asks for permission, you must allow Apache and MySQL access to "Private Networks."**
3. **The Document Root: During installation, note the location of the `www` folder.**

- Windows: `C:\Ampps\www`
- macOS: `/Applications/AMPPS/www`

Step 2: The Dashboard Control

Open the AMPPS Application. You will see a list of services. For our work, Apache and MySQL must show a green status.

- **If Apache is red: Another program (like Skype or Zoom) is likely "stealing" Port 80.**
- **The Fix: Close those apps and restart AMPPS.**

2.4 The "Under the Hood" Configuration Files

A professional IT student does not just click "Start"; they understand the configuration files that govern the server.

The Apache Heart: `httpd.conf`

Apache's behavior is dictated by a text file called `httpd.conf`. Inside this file, you will find:

- **Listen 80: Tells Apache to "listen" for requests on Port 80.**
- **DocumentRoot: The physical path on your hard drive where your website files live.**
- **DirectoryIndex: Tells Apache which file to show first (usually `index.php` or `index.html`).**

The PHP Brain: `php.ini`

This file controls the limits of the PHP engine.

- **memory_limit: How much RAM a single script can use (e.g., `128M`).**
- **upload_max_filesize: The maximum size of a file a user can upload to your site.**
- **display_errors: In development, we set this to `On` so we can see our mistakes. In a real website, we set it to `Off` so hackers can't see our code's inner workings.**

2.5 Creating Your First "Live" Script

Let's verify the stack is working by creating a diagnostic script.

1. **Open your text editor (like VS Code or Notepad++).**
2. **Type the following code exactly:**

```
<?php

 // This function outputs the entire configuration of your server

 phpinfo();
```

```
?>
```

3. Save this file as `test.php` inside your `www` folder.
4. Open your browser and type: `http://localhost/test.php`

Logic Breakdown:

- `<?php`: This is the Opening Tag. It tells the server: "Stop treating this as HTML and start processing this as PHP code."
- `phpinfo();`: This is a Built-in Function. It tells the PHP engine to generate a massive table showing every detail of your server setup.
- `?>`: This is the Closing Tag. It tells the server the PHP logic is finished.

2.6 Security: The "Port 80 War"

As mentioned in the previous version, Port 80 is the standard "door" for web traffic. Only one program can use a door at a time.

Common Beginner Mistake: Running "World Wide Web Publishing Service" (built into Windows) while trying to run Apache.

The Solution: Open "Services" in Windows, find that service, and "Stop" it. This clears the path for Apache to take control of the traffic.

2.7 Practical Mini-Lab: Customizing PHP

1. Open the AMPPS control panel.
2. Click the "PHP" icon and select "Configuration" (this opens `php.ini`).
3. Search for `max_execution_time`. It is likely set to `30`.
4. Change it to `60` and save the file.
5. Important: Restart Apache for the change to take effect. You have just doubled the time a script is allowed to run before the server kills it.

2.8 Chapter 2 Review Questions

1. What does the acronym LAMP stand for?
2. Explain the purpose of a "Development Environment" vs. a "Production Environment."
3. What is the significance of the IP address `127.0.0.1`?
4. In which configuration file would you change the maximum allowed file upload size?
5. What happens if you try to run Apache while Port 80 is being used by another application?
6. What is a "Document Root" in Apache?

7. **Why is it dangerous to have `display_errors` set to `On` on a public, live website?**
8. **What is the purpose of the `phpinfo()` function?**
9. **If your `www` folder is located at `C:\Ampps\www`, and you save a file called `my-site.php` there, what URL do you type in your browser to see it?**
10. **Describe the role of the "Three-Way Handshake" in establishing a server connection.**

MODULE 2: PHP PROGRAMMING

The Brain of the Web Application

Chapter 3: Introduction to PHP

3.1 Technical Preface: Why PHP?

In our previous module, we learned that a server sends HTML to a browser. But HTML is "dumb"—it cannot think. If you want to show the current time, or check if a password is correct, you need a language that can perform logic.

PHP (Hypertext Preprocessor) is that language. It is a Server-Side language, meaning the "thinking" happens on the computer in the data center, not on your phone or laptop. By the time the data reaches you, the PHP has already finished its work and turned into standard HTML. This is why you can never "View Source" on a website and see the PHP code; you only see the result of its thoughts.

3.2 The "Lego Analogy" of Syntax

If HTML is the physical brick of a building, PHP is the instruction manual that tells the bricks where to go. Every PHP script follows a strict "Grammar" or Syntax.

The Opening and Closing Tags

A PHP code block must always be wrapped in these tags:

- **`<?php` — This tells the server: "Start the engine. Everything following this is logic."**
- **`?>` — This tells the server: "Stop the engine. Go back to treating text as plain HTML."**

3.3 Variables: The Memory Buckets

The most important concept in programming is the Variable. Imagine you have a physical bucket. You put a label on the bucket called "Price," and inside the bucket, you put a piece of paper that says "10."

In PHP, a variable always starts with a Dollar Sign (`$`).

```
$user_age = 25;
```

Code Anatomy:

1. **$: The symbol that identifies a variable.**
2. **user_age: The name of the "bucket." Names must not have spaces.**
3. **=: The Assignment Operator. It means "Put the value on the right into the bucket on the left."**
4. **25: The value.**
5. **;: The Semicolon. This is the most important character. It acts like the period at the end of a sentence. If you forget it, the engine crashes.**

3.4 Data Types: What's in the Bucket?

PHP is "Loosely Typed," meaning it tries to guess what kind of data you are using. There are four primary types you must know:

1. **Integer: Whole numbers (e.g., 5, -10, 1000).**
2. **Float: Numbers with decimals (e.g., 10.99, 3.14).**
3. **String: Text. These must be wrapped in quotes (e.g., "Hello Student").**
4. **Boolean: A simple switch. It can only be true or false.**

Chapter 4: Expressions and Control Flow

4.1 Making Decisions (The if Statement)

Control flow is how we give the "Brain" its personality. The most common tool is the if statement.

Analogy: "If it is raining, take an umbrella. Otherwise, wear sunglasses."

```
<?php

 $weather = "raining";



 if ($weather == "raining") {

    echo "Take an umbrella!";

 } else {

    echo "Wear sunglasses!";

 }

?>
```

Step-by-Step Logic Breakdown:

- `if ($weather == "raining")`: We use a double equals (`==`) to ask a question. "Does the bucket match this text?"
- `{` and `}`: These Curly Brackets create a "Room." Everything inside these brackets only happens if the answer to the question is "Yes."
- `echo`: This is the command to "print" or "speak" text to the screen.
- `else`: This handles the "No" answer.

4.2 Loops: The Power of Repetition

Computers are great at doing boring things millions of times without getting tired. We use Loops for this.

The `while` Loop

A `while` loop continues to run as long as a condition is true.

```
<?php

 $count = 1;



 while ($count <= 5) {

    echo "This is line number " . $count . "<br>";

    $count = $count + 1;

 }

?>
```

Why the last line matters: If we didn't add 1 to the count, it would stay at 1 forever, and the computer would never stop printing. This is called an Infinite Loop, and it is a common beginner mistake that can "freeze" your browser.

Chapter 5: PHP Functions and Objects

5.1 Functions: The Reusable Recipe

A Function is a saved block of code that you can give a name to. Instead of writing 10

lines of code to calculate tax every time, you write it once in a function and "Call" it by name.

```
function calculate_tax($price) {
    return $price * 0.15;
}

echo calculate_tax(100); // Outputs 15
```

5.2 Objects: The Car Blueprint

As you become a professional, you will use Object-Oriented Programming (OOP).

- **The Class: This is a "Blueprint." It describes what a "User" is (they have a name, an email, and a password).**
- **The Object: This is the actual "User" created from the blueprint (e.g., a user named "John").**

Chapter 6: PHP Arrays

6.1 The "Egg Carton" Analogy

An Array is a variable that can hold many values at once. Think of it like an egg carton. The carton is one object, but it has 12 individual slots.

- **Numeric Array: Slots are numbered `0, 1, 2, 3...` (Computers always start counting at zero!).**
- **Associative Array: Slots have names. Instead of slot `0`, you have a slot named `"username"`.**

Chapter 7: Practical PHP (Files and Scope)

7.1 Variable Scope

Variables have a "Life Span."

- **Local Scope: A variable created inside a function "dies" the moment the function is finished.**
- **Global Scope: A variable created at the top of your script "lives" as long as the page is loading.**

7.2 Including Files

Professional developers don't put all their code in one giant file. They use `include` or `require`.

- **`include 'header.php';` — This "glues" the code from `header.php` into your current page. This is how websites keep their menus consistent across every page.**

7.3 Common Beginner Pitfalls

1. **Missing Semicolons: 90% of your errors will be a missing `;`.**
2. **The Dollar Sign: Forgetting the `$` before a variable name.**
3. **Quotes around Numbers: Writing `$age = "25";` makes it a String (text), not an Integer (number). You cannot do math on text!**

7.4 Chapter Review Questions

1. **What is the difference between a Server-Side language and a Client-Side language?**
2. **Why must every PHP variable start with a `$`?**
3. **What happens if you forget a semicolon at the end of a line?**
4. **Explain the difference between `==` (Assignment) and `==` (Comparison).**
5. **What is an "Infinite Loop"?**
6. **Why do we start counting at `0` in an array?**
7. **What is the difference between `include` and `require`?**
8. **Write a function that takes a name as input and echoes "Hello, [Name]".**
9. **What is a "Boolean" data type?**
10. **Using an analogy, explain what a "Class" is in OOP.**

MODULE 3: MYSQL DATABASES

The Data Vault: Mastering Permanent Storage

Chapter 8: Introduction to MySQL

8.1 Technical Preface: Why Databases?

In the previous modules, our PHP scripts were "forgetful." Once a script finishes execution, every variable—every $name, every $price—is wiped from the computer's RAM. To build a system that remembers users, orders, or posts, we need a **Database**.

MySQL is a **Relational Database Management System (RDBMS)**. It uses **SQL (Structured Query Language)**, a standardized language that allows humans to "talk" to data using logical, English-like commands.

8.2 The Relational Model: Tables, Rows, and Columns

Think of a MySQL database not as a single file, but as a collection of interconnected **Spreadsheets**.

- **The Database:** The entire "File" or "Container" (e.g., OnlineStore).
- **The Table:** A specific category inside that container (e.g., Customers).
- **The Column (Field):** A specific type of data (e.g., Email_Address).
- **The Row (Record):** A single horizontal entry (e.g., "John Doe, john@email.com").

8.3 Data Types: Choosing the Right Bucket

Just like PHP variables, MySQL columns must be told exactly what kind of data they will hold. This is crucial for performance and storage efficiency.

6. **INT:** Whole numbers. Used for IDs, age, or quantities.
7. **VARCHAR(length):** Variable-length text (e.g., VARCHAR(100) for a name). It only uses as much space as the text actually takes up.
8. **TEXT:** For large blocks of text, like a blog post body.
9. **DATE:** Formatted as YYYY-MM-DD.

Chapter 9: Database Design and Normalization

9.1 The Problem of Redundancy

A common beginner mistake is putting all data into one giant table. Imagine a table where you store a customer's name next to every item they buy. If "John Smith" buys 50 items, his name is written 50 times. If he changes his name, you have to update 50 rows. This leads to **Data Corruption**.

9.2 Normalization (1NF, 2NF, 3NF)

Normalization is the process of organizing data into multiple related tables to eliminate redundancy.

- **1NF (First Normal Form):** Every cell must contain only one value. No lists inside a cell.
- **2NF (Second Normal Form):** Move redundant data to new tables and connect them using a **Primary Key**.
- **3NF (Third Normal Form):** Every column must only depend on the Primary Key.

Primary Keys and Foreign Keys

- **Primary Key (PK):** A unique ID (usually an INT AUTO_INCREMENT) that identifies a specific row. No two rows can have the same PK.
- **Foreign Key (FK):** A column in one table that points to the Primary Key in another table, creating a "Relationship."

9.3 ACID Properties: The Guarantee of Safety

Professional databases follow **ACID** rules to ensure your data is never lost:

4. **Atomicity:** An operation is "all or nothing." If a power outage happens mid-transaction, the database rolls back to the start.
5. **Consistency:** Data must follow all rules (e.g., a "Price" cannot be a word).
6. **Isolation:** Transactions don't interfere with each other.
7. **Durability:** Once a change is saved, it stays saved, even if the system crashes.

Chapter 10: Using MySQL with PHP

10.1 The mysqli Extension

PHP doesn't "speak" SQL natively; it uses an interface called mysqli (MySQL Improved). We treat the database as an **Object**.

10.2 CRUD: The Four Pillars of Data

Every application does four things: **C**reate, **R**ead, **U**pdate, and **D**elete.

Security Warning: SQL Injection

If you put a user's typed name directly into a query, a hacker can type a command like '; DROP TABLE users; -- to delete your database.

The Solution: Placeholders (?). We send the command first, and the data separately. This is called a **Prepared Statement**.

10.3 Step-by-Step Coding Walkthrough

```
<?php
 // 1. Establish Connection
 $conn = new mysqli("localhost", "root", "", "my_database");

 // 2. Prepare the Query with Placeholders
 $stmt = $conn->prepare("SELECT email FROM users WHERE username = ?");

 // 3. Bind the Data (s = string)
 $username = "john_doe";
 $stmt->bind_param("s", $username);

 // 4. Execute and Fetch
 $stmt->execute();
 $result = $stmt->get_result();
 $user = $result->fetch_assoc();

 echo "The email is: " . $user['email'];
?>
```

Line-by-Line Breakdown:

6. new mysqli(...): Opens the "Door" to the database.
7. prepare(...): Sends the "Plan" to MySQL without the dangerous data.
8. bind_param("s", ...): Hands the data to the engine securely. "s" tells MySQL the data is a string.
9. fetch_assoc(): Turns the database row into a PHP **Associative Array**.

10.4 Common Beginner Pitfalls

- **The "Case" Mistake:** SQL keywords like SELECT are not case-sensitive, but table names often *are* on Linux servers. Always use consistent casing.
- **Connection Errors:** Forgetting to start the MySQL service in your AMPPS/XAMPP control panel.
- **Primary Key Forgetfulness:** Forgetting to set an id column to AUTO_INCREMENT, forcing you to manually figure out the next number every time.

10.5 Chapter Review Questions

11. What does the "R" in RDBMS stand for, and why is it important?
12. Explain the difference between a VARCHAR and a TEXT data type.

13. Why should you never store a customer’s name multiple times in an orders table?
14. Define "Normalization" in your own words.
15. What is a "Primary Key," and how does it differ from a "Foreign Key"?
16. Explain the "A" (Atomicity) in the ACID model.
17. What is "SQL Injection," and how does it work?
18. What is the purpose of a "Prepared Statement"?
19. Describe what the fetch_assoc() function does to a database result.
20. Write the SQL command to create a table named books with columns for id, title, and author.

MODULE 4: FORMS, SESSIONS & SECURITY

Bridging the Gap Between Client and Server

Chapter 11: Interacting with the User via Forms

11.1 Technical Preface: The Gateway of Input

If the database is the "Vault," then HTML Forms are the "Front Desk" where users submit information. Whether it is a search bar, a login screen, or a profile editor, forms are the primary way users send data to your PHP scripts.

11.2 Anatomy of an HTML Form

A form is a container in HTML that defines how data should be packaged and where it should be sent.

```
<form action="process.php" method="POST">
  <label for="username">Username:</label>
  <input type="text" name="username" id="username">
  <input type="submit" value="Log In">
</form>
```

Step-by-Step Logic Breakdown:

1. **`action="process.php"`: This is the destination. It tells the browser: "When the user clicks submit, send all the data to this specific PHP file."**
2. **`method="POST"`: This is the transport method. There are two main types:**
    - **GET: Attaches data to the URL (e.g., `process.php?username=John`). Visible to the user, bookmarkable, but insecure for passwords.**
    - **POST: Sends data "under the hood" in the HTTP body. Private and capable of sending large amounts of data.**
3. **`name="username"`: This is the most important attribute for PHP. It creates the "Key" that PHP will use to find the data in the `$_POST` array.**

11.3 Processing Input in PHP

When the form is submitted, PHP automatically creates a "Superglobal" array. If you used `method="POST"`, the data lives in `$_POST`.

```
<?php
  // Checking if the data exists before using it
  if (isset($_POST['username'])) {
      $user = $_POST['username'];
      echo "Welcome back, " . htmlspecialchars($user);
  }
?>
```

Security Note: We use `htmlspecialchars()` to prevent XSS (Cross-Site Scripting). It turns characters like `<` into `<`, stopping hackers from injecting malicious scripts into your page.

Chapter 12: Cookies, Sessions, and Authentication

12.1 Technical Preface: Solving the "Stateless" Problem

As learned in Chapter 1, HTTP is Stateless. This means that when you click a link to go from `page1.php` to `page2.php`, the server has no idea you are the same person. To solve this, we use two "Memory" tools: Cookies and Sessions.

12.2 Cookies: The Client's ID Card

A Cookie is a small text file that the server asks the browser to save on the user's hard drive.

- **The Mechanic: The server sends a `Set-Cookie` header. Every time the browser visits that site again, it automatically attaches that cookie to the request.**
- **The Risk: Users can see and edit cookies. Never store sensitive data like passwords in a cookie.**

12.3 Sessions: The Server's Filing Cabinet

A Session is far more secure. The data stays on the server. The only thing the user gets

is a "Session ID" (usually stored in a cookie called `PHPSESSID`).

The Analogy: A Cookie is like a loyalty card you carry in your wallet. A Session is like a tab at a bar—the bartender keeps the record, and you just show your face (the ID) to access it.

Implementing a Secure Session

```
<?php
 // Must be the very first line of the file!
 session_start();


 // Storing data
 $_SESSION['user_id'] = 42;
 $_SESSION['is_logged_in'] = true;


 // Checking data on another page
 if ($_SESSION['is_logged_in']) {
    echo "Access Granted.";
 } else {
    die("Please log in first.");
 }
?>
```

12.4 Security: The "Big Three" Threats

1. **Session Hijacking: A hacker steals a user's Session ID cookie.**
    - ***Mitigation*: Use `session_regenerate_id()` during login and force HTTPS.**
2. **SQL Injection: Hackers use form inputs to run database commands.**
    - ***Mitigation*: Use Prepared Statements (see Chapter 10).**
3. **Cross-Site Request Forgery (CSRF): A malicious site tricks a logged-in user into**

clicking a link that performs an action on your site (like "Delete Account").

- ***Mitigation*: Use "CSRF Tokens"—unique hidden codes in forms that PHP verifies.**

12.5 Common Beginner Pitfalls

- **"Headers Already Sent": This error happens if you try to use `session_start()` or `header()` after you have already echoed text or HTML. PHP must send the "Envelope" (Headers) before it sends the "Letter" (HTML).**
- **The `name` vs `id` Confusion: In HTML, `id` is for CSS/JavaScript, but `name` is what PHP uses to grab data. If you forget the `name` attribute, your `$_POST` array will be empty!**

12.6 Chapter Review Questions

1. **Compare and contrast the `GET` and `POST` methods. When is `GET` appropriate?**
2. **What is a "Superglobal" array in PHP? Give two examples.**
3. **Explain the purpose of the `action` attribute in an HTML form tag.**
4. **Why is it dangerous to store a user's "Admin Status" in a Cookie?**
5. **What must be the very first line of any PHP file that uses sessions?**
6. **How does `htmlspecialchars()` protect a website?**
7. **Describe the process of "Session Hijacking."**
8. **What does `session_destroy()` do, and when should it be used?**
9. **Why does a browser automatically send cookies back to a server?**
10. **In a login form, why is it better to use `POST` than `GET`?**

MODULE 5: JAVASCRIPT FUNDAMENTALS

The Language of the Browser

Chapter 13: Introduction to Client-Side Scripting

13.1 Technical Preface: Why JavaScript?

Until now, we have used PHP to think for our website. But PHP is slow for small tasks because it requires a "Round Trip"—the request must travel thousands of miles to the server and back just to change a color or check if a box is ticked.

JavaScript is a Client-Side language. It lives and breathes inside the user's browser (Chrome, Safari, Firefox). It allows us to manipulate the page *after* it has finished loading from the server.

13.2 The Fundamental Syntax

While PHP uses `$variable`, JavaScript uses the keywords `let`, `const`, or `var`.

- **`let`: For variables that will change (e.g., a score in a game).**
- **`const`: For values that stay the same (e.g., a mathematical constant).**

```
// A simple JavaScript alert

let userName = "Alex";

console.log("Hello " + userName);
```

Step-by-Step Logic Breakdown:

1. **`let userName`: Creates a "bucket" in the browser's memory.**
2. **`console.log`: This is the JavaScript equivalent of `echo`. It prints data to the browser's "Console" (hidden from the user, but visible to developers via F12).**
3. **Semicolons: Just like PHP, semicolons end a statement.**

Chapter 14: JavaScript Expressions and Control Flow

14.1 Logic in the Browser

JavaScript uses almost identical logic to PHP for `if` statements and loops, but the way it handles types is different.

The Identity Operator (`===`): In JavaScript, always use three equals signs. This checks if the value *and* the type are the same (e.g., it ensures the number `5` is not mistaken for the text `"5"`).

```
let age = 18;

if (age === 18) {
    alert("You just became an adult!");
}
```

Chapter 15: Functions, Objects, and Scope

15.1 Functions: The Reusable Logic

JavaScript functions can be triggered by "Events," such as a user clicking a button.

```
function sayHello() {
    alert("The button was clicked!");
}
```

15.2 Global vs. Local Scope

- **Global: A variable declared outside a function is available to every script on the page.**
- **Local: A variable declared inside a function is "born" and "dies" within those curly brackets `{ }`.**

Chapter 16: The DOM (Document Object Model)

16.1 Technical Preface: How JS "Sees" the Page

When a browser loads HTML, it converts it into a "Tree" structure called the DOM. JavaScript uses this tree to find and change elements.

The Analogy: If the HTML is a physical house, the DOM is the electrical blueprint. JavaScript is the hand that flips the switches to turn on the lights or change the wall color.

Manipulating an Element

```
<p id="message">Old Text</p>



<script>

 // Find the element by its ID and change its content

 let element = document.getElementById("message");

 element.innerHTML = "New Interactive Text!";

 element.style.color = "blue";

</script>
```

Line-by-Line Breakdown:

1. `document`: **The root of the entire webpage.**
2. `getElementById("message")`: **JavaScript "reaches out" and grabs the specific paragraph.**
3. `.innerHTML`: **Changes the text inside the tags.**
4. `.style.color`: **Changes the CSS directly through code.**

16.2 Common Beginner Pitfalls

- **The "Null" Error: If you try to grab an element with JavaScript *before* the HTML has finished loading, you will get a "null" error. Always place your `<script>` tags at the bottom of the `<body>`.**
- **Case Sensitivity: JavaScript is strictly case-sensitive. `myVariable` and `myvariable` are two completely different things.**
- **Alert Overuse: Beginners love `alert()`, but it freezes the entire browser. Professionals use `console.log()` for testing.**

16.3 Chapter Review Questions

1. **Explain the difference between Server-Side (PHP) and Client-Side (JavaScript) execution.**
2. **What is the purpose of the `const` keyword?**
3. **How do you see the output of `console.log()` in your browser?**
4. **What does the "DOM" stand for?**
5. **Why is it better to use `===` instead of `==` in JavaScript?**
6. **Explain the "Event" concept (e.g., `onclick`).**
7. **What happens to a local variable once a function finishes running?**
8. **How can JavaScript change the CSS of an element?**
9. **Why should `<script>` tags usually be placed at the end of an HTML file?**
10. **Write a JavaScript function that changes the background color of the page to "red."**

MODULE 6: ASYNCHRONOUS WEB & AJAX

The Silent Conversation: Updating Without Refreshing

Chapter 17: Asynchronous Communication (AJAX)

17.1 Technical Preface: What is AJAX?

In the previous modules, we learned that web interaction usually works like this: You click a button, the screen goes white for a second, and a whole new page loads. This is called Synchronous communication.

AJAX stands for Asynchronous JavaScript And XML. It is the "Holy Grail" of user experience. It allows JavaScript to send a request to a PHP script in the background. When the PHP script finishes its work (like checking a database), it sends the answer back to the JavaScript, which then updates just a small piece of the page.

The Analogy:

- **Synchronous (Non-AJAX): You go to a restaurant, order food, and have to leave the building and come back in just to see if your appetizer is ready.**
- **Asynchronous (AJAX): You sit at the table, and the waiter (the background process) brings you items as they are ready while you continue your conversation.**

17.2 The Mechanics of the XMLHttpRequest

While modern developers often use newer tools (like `fetch`), it is vital for an IT student to understand the original engine: the XMLHttpRequest (XHR) object.

The 4-Step AJAX Cycle:

1. **The Trigger: A user event (like typing in a search box) starts a JavaScript function.**
2. **The Request: JavaScript creates an XHR object and sends it to the server.**
3. **The Server Logic: A PHP script receives the request, talks to the MySQL database, and `echo`es a result.**
4. **The Update: JavaScript receives that result and uses the DOM (from Chapter 16) to inject the data into the page.**

17.3 Conceptual Walkthrough: Real-Time Username Checker

Imagine a signup form where you want to tell the user "Username Taken" the moment they finish typing, without them hitting "Submit."

The PHP (backend_check.php):

```
<?php
  // This script just checks if a name is 'Admin'
  $name = $_GET['user'] ?? '';
  if ($name === 'Admin') {
      echo "Taken";
  } else {
      echo "Available";
  }
?>
```

The JavaScript (frontend.js):

```
function checkUsername(str) {
    if (str.length == 0) return; // Don't check empty strings

    // 1. Create the request object
    let xmlhttp = new XMLHttpRequest();

    // 2. Define what to do when the answer comes back
    xmlhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("status").innerHTML = this.responseText;
```

```
    }
  };

  // 3. Open the connection and send
  xmlhttp.open("GET", "backend_check.php?user=" + str, true);
  xmlhttp.send();
}
```

17.4 Deep Dive: JSON (The Modern Data Language)

Originally, AJAX used XML (a clunky, tag-heavy format). Today, we use JSON (JavaScript Object Notation). It is much lighter and looks exactly like a JavaScript object.

Example JSON:

```
{"status": "success", "userId": 42}
```

In PHP, we turn an array into JSON using `json_encode($array)`. In JavaScript, we turn that text back into an object using `JSON.parse(text)`.

17.5 Common Beginner Pitfalls

- **The "Same-Origin" Policy: For security reasons, AJAX cannot talk to a different website (e.g., your site cannot AJAX request data from `facebook.com`) unless that site explicitly allows it. This is a common source of "CORS Errors."**
- **Forgetting `readyState == 4`: The AJAX object goes through 5 stages (0 to 4). If you try to read the data at stage 2, it will be empty. Stage 4 means "Complete."**
- **The Browser Cache: Sometimes browsers "remember" an AJAX response and don't actually ask the server again. Adding a random number to the URL (like `?t=12345`) prevents this.**

17.6 Chapter Review Questions

1. **What does the "A" in AJAX stand for, and why is it important for user experience?**
2. **Describe the "Round-Trip" difference between a traditional page load and an AJAX request.**
3. **What is the `XMLHttpRequest` object?**
4. **In the `onreadystatechange` function, what does `status == 200` mean?**
5. **Why is JSON preferred over XML in modern web development?**

6. How does AJAX improve the speed of a web application?
7. Explain the "Same-Origin Policy" in your own words.
8. What PHP function is used to send data back to an AJAX request in JSON format?
9. Give three real-world examples of AJAX being used in apps you use daily.
10. If an AJAX request returns a "500" error, where should you look for the bug—the JavaScript or the PHP?

MODULE 7: CSS & LAYOUT SYSTEMS

The Art of Structure: Styling the Modern Web

Chapter 18: Cascading Style Sheets (CSS) Selectors and Rules

18.1 Technical Preface: The Separation of Concerns

In the early days of the web, developers used HTML tags like `<font>` and `<center>` to style pages. This was a nightmare to maintain. The solution was CSS.

The Core Principle: HTML handles the Content (the "What"), and CSS handles the Presentation (the "How it looks"). This allows you to change the entire look of a 1,000-page website by editing just one CSS file.

18.2 The Anatomy of a CSS Rule

A CSS rule consists of a Selector and a Declaration Block.

```
p {
  color: blue;
  font-size: 16px;
}
```

Step-by-Step Logic Breakdown:

1. **`p` (The Selector): Tells the browser: "Find all paragraph tags on the page."**

2. **{ } (The Block): Contains the styles to apply to those paragraphs.**
3. **color: blue;: A Property (color) and a Value (blue).**
4. **The Semicolon (;): Just like PHP and JS, every declaration must end with a semicolon.**

18.3 The Three Types of Selectors

1. **Element Selector: Targets tags directly (e.g., h1, div).**
2. **Class Selector (.): Targets specific elements you have labeled. (e.g., .highlight { background: yellow; }). You can use a class on many elements.**
3. **ID Selector (#): Targets one unique element. (e.g., #main-header). An ID should only be used once per page.**

Chapter 19: The CSS Box Model and Positioning

19.1 Technical Preface: Everything is a Box

To master CSS, you must understand that the browser sees every single element (images, text, links) as a rectangular box.

19.2 Anatomy of the Box Model

From the inside out, every box consists of:

1. **Content: The actual text or image.**
2. **Padding: The invisible "cushion" inside the box, pushing the content away from the border.**
3. **Border: The line surrounding the padding and content.**
4. **Margin: The "social distancing" space outside the box, pushing other elements away.**

The "Standard Box Model" Trap: By default, if you set a box to width: 200px and add 10px of padding, the box actually becomes 220px wide. To fix this and make math easier, professionals use:

```
box-sizing: border-box;
```

19.3 Positioning: Placing the Boxes

- **Static: The default. Boxes flow one after another.**
- **Relative: Moves the box slightly from its original spot without affecting others.**
- **Absolute: Places a box exactly where you want inside a parent container.**
- **Fixed: Glues the box to the screen (like a navigation bar that stays as you scroll).**

Chapter 20: Advanced Layouts (Flexbox and Grid)

20.1 Technical Preface: Moving Beyond Floats

In the past, making columns was difficult. Today, we use Flexbox for 1-dimensional layouts (rows OR columns) and CSS Grid for 2-dimensional layouts (rows AND columns).

The Power of Flexbox

By simply saying `display: flex;` on a container, you can perfectly center items or space them out evenly.

```
.container {
  display: flex;
  justify-content: space-around;
  align-items: center;
}
```

20.2 Responsive Design: Media Queries

A website must work on a 30-inch monitor and a 5-inch phone. We use Media Queries to tell the browser: "If the screen is smaller than 600px, change the layout."

```
@media (max-width: 600px) {
  body {
    background-color: lightblue;
    flex-direction: column; /* Stack items vertically on mobile */
  }
}
```

20.3 Common Beginner Pitfalls

- **The "Specificity" War: If you have one rule saying text is red and another saying it's blue, CSS follows a hierarchy. ID beats Class, and Class beats Element.**
- **Forgetting the "Unit": Writing `margin: 20` instead of `margin: 20px`. Without the unit, the browser ignores the rule.**
- **The "Invisible" Box: Sometimes a box disappears because it has a `height: 0`. If a box has no content and no set height, you won't see it!**

20.4 Chapter Review Questions

1. **Explain the "Separation of Concerns" between HTML and CSS.**
2. **What is the difference between a `padding` and a `margin`?**
3. **How does `box-sizing: border-box` change the way element width is calculated?**
4. **When should you use an ID selector instead of a Class selector?**
5. **What does the "Cascade" in Cascading Style Sheets actually mean?**
6. **Describe the four layers of the CSS Box Model from inside to outside.**
7. **What is the purpose of a "Media Query" in modern web design?**
8. **How do you center an item horizontally using Flexbox?**
9. **Explain the difference between `position: absolute` and `position: fixed`.**
10. **Write a CSS rule that turns all `h2` elements green and gives them a `20px` bottom margin.**

MODULE 8: JQUERY & MOBILE

Streamlining Logic for the Modern Device

Chapter 21: Introduction to jQuery

21.1 Technical Preface: Write Less, Do More

JavaScript is powerful, but it can be verbose. To perform a simple task like hiding a box, you might need to write several lines of "Vanilla" JavaScript. In 2006, John Resig released jQuery, a library designed to simplify DOM manipulation, event handling, and AJAX.

The jQuery Philosophy: It wraps complex JavaScript tasks into simple methods. Instead of writing long document selectors, jQuery uses the same syntax as CSS to "grab" elements.

21.2 The jQuery Syntax: The Magic of $

In jQuery, the dollar sign ($) is a shortcut for the jQuery() function. A standard jQuery statement follows this pattern: $(selector).action().

```
// Vanilla JavaScript

document.getElementById("myButton").addEventListener("click", function() {

    document.getElementById("box").style.display = "none";

});



// jQuery Equivalent

$("#myButton").click(function() {

    $("#box").hide();

});
```

Step-by-Step Logic Breakdown:

1. **$("#myButton"): Uses the CSS ID selector to find the button.**
2. **.click(): A built-in "Event Method" that waits for a user click.**

3. **.hide(): A "Method" that handles all the complex CSS logic to make an element disappear.**

21.3 DOM Manipulation and Effects

jQuery excels at "Effects"—pre-written animations that would take hours to code in raw CSS or JS.

- **.fadeIn() / .fadeOut(): Gradually changes opacity.**
- **.slideUp() / .slideDown(): Creates a drawer-like opening effect.**
- **.css(): Changes multiple style properties at once using a JavaScript object.**

Chapter 22: jQuery Mobile and Mobile-First Design

22.1 Technical Preface: The Mobile Web App

A mobile website is not just a desktop site that is smaller; it is an experience designed for "Thumbing" rather than "Clicking." jQuery Mobile is a framework built on top of jQuery that provides a unified user interface (UI) system for all smartphone platforms.

22.2 Touch-Friendly Components

jQuery Mobile uses Data Attributes (data-role) to turn standard HTML tags into mobile-optimized components.

```
<!-- A mobile-styled header and button -->

<div data-role="header">

  <h1>My App</h1>

</div>



<div data-role="main" class="ui-content">

  <a href="#page2" data-role="button" data-icon="gear">Settings</a>

</div>
```

Architectural Insight: jQuery Mobile uses an AJAX-based navigation system. When you click a link, it doesn't load a whole new page. Instead, it pulls the content of the next

page in the background and uses a smooth transition (like a slide) to show it.

22.3 Responsive Tables and Grids

Because mobile screens are narrow, jQuery Mobile provides "Reflow" tables. On a desktop, it looks like a normal table. On a phone, it automatically stacks the columns vertically so the user doesn't have to scroll sideways.

22.4 Common Beginner Pitfalls

- **Forgetting the Library: jQuery is not built into the browser. If you don't include the `<script>` link to the jQuery file, your `$` code will throw a "ReferenceError."**
- **The Document Ready Wrapper: JavaScript often tries to run before the HTML is finished loading. In jQuery, we always wrap our code in `$(document).ready(function() { ... });` to ensure the "stage is set" before the "actors" (our code) start performing.**
- **Version Conflicts: Using a very old version of jQuery with a very new plugin can cause the site to break. Always check compatibility.**

22.5 Chapter Review Questions

1. **What is the main goal of the jQuery library?**
2. **Explain the meaning of the `$` symbol in a jQuery statement.**
3. **How does jQuery's selector system differ from Vanilla JavaScript's `getElementById`?**
4. **What is the purpose of the `$(document).ready()` function?**
5. **List three jQuery "Effects" and describe what they do visually.**
6. **How does jQuery Mobile differ from standard jQuery?**
7. **What is a "Data Attribute" (e.g., `data-role`), and why is it used in mobile frameworks?**
8. **Explain why AJAX-based navigation is preferred for mobile web apps.**
9. **What is "Mobile-First" design?**
10. **Write a jQuery script that changes the text of all `<h1>` tags to "Hello World" when a button with the ID `btn` is clicked.**

MODULE 9: HTML5 & ADVANCED FEATURES

The Modern Web Platform: Power Beyond the Document

Chapter 23: The Canvas API and Multimedia

23.1 Technical Preface: From Document to Application

Originally, HTML was designed to share scientific papers. HTML5 changed the "Internet's DNA," turning the browser into a full-fledged application platform. Two of its most powerful features are the `<canvas>` element and native multimedia support.

23.2 Drawing with the Canvas

The `<canvas>` is a resolution-dependent bitmap canvas which can be used for rendering graphs, game graphics, or other visual images on the fly via JavaScript.

The "Artist" Analogy:

HTML provides the physical canvas (the `<canvas>` tag), but JavaScript is the "Painter" that actually holds the brush and applies the color.

```
// Step-by-step Canvas drawing

let canvas = document.getElementById("myCanvas");

let ctx = canvas.getContext("2d"); // The 2D "Pen"



ctx.fillStyle = "red"; // Pick up red paint

ctx.fillRect(20, 20, 150, 100); // Draw a rectangle (x, y, width, height)
```

23.3 Native Audio and Video

Before HTML5, you needed a "Plugin" (like Flash) to play video. This was slow and insecure. HTML5 introduced the `<video>` and `<audio>` tags, which allow the browser to

play media natively.

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  Your browser does not support the video tag.
</video>
```

Chapter 24: Geolocation and Local Storage

24.1 Geolocation: Where in the World?

The Geolocation API allows the user to share their physical location with your web application. This is used for maps, localized weather, or finding the nearest store.

Privacy Note: For security, the browser will always ask the user for permission before sharing their location. If the user says "No," your code must handle that "Denied" state gracefully.

24.2 Local Storage: The Browser's Internal Database

In Module 4, we learned about Sessions and Cookies. Local Storage is a modern upgrade. It allows you to save large amounts of data (up to 5MB or more) directly in the browser with no expiration date.

- **Session Storage: Data is lost when the tab is closed.**
- **Local Storage: Data stays even if the computer is restarted.**

```
// Saving data
localStorage.setItem("username", "JohnDoe");

// Retrieving data
let name = localStorage.getItem("username");
```

Chapter 25: Web Workers and Offline Capabilities

25.1 Web Workers: Multi-Tasking in the Browser

Normally, JavaScript is "Single-Threaded." This means if you run a very complex calculation, the whole webpage freezes until the calculation is done. Web Workers allow you to run heavy logic in a "Background Thread," keeping the user interface smooth and responsive.

25.2 The Application Cache and Service Workers

Modern HTML5 features allow websites to work Offline. A "Service Worker" acts as a proxy between the browser and the network, allowing you to serve cached files when there is no internet connection.

Chapter 26: Drag and Drop & Microdata

26.1 Drag and Drop API

HTML5 makes any element "draggable." This allows for intuitive interfaces like file upload zones or rearranging items in a list. You define which elements can be dragged and where they can be "dropped" using event listeners.

26.2 Microdata and SEO

Microdata is a way to label the content of your page so that search engines (like Google) understand what it is. For example, instead of just text, you can label a block of code as a "Recipe," "Product," or "Event," allowing search engines to show "Rich Snippets" in search results.

26.3 Common Beginner Pitfalls

- **Canvas Coordinates: Beginners often get confused by the coordinate system. In `<canvas>`, `(0,0)` is the Top-Left corner. As `Y` increases, you move *down*.**
- **Local Storage Types: Local Storage only stores Strings. If you want to store an object or an array, you must use `JSON.stringify()` before saving and `JSON.parse()` after retrieving.**
- **HTTPS Requirement: Most HTML5 "Power Features" (like Geolocation and Web Workers) will only work if your website is running on a secure HTTPS connection.**

26.4 Chapter Review Questions

1. **How does the `<canvas>` element differ from a standard `<img>` tag?**

2. **Explain the role of the "Context" (e.g., `getContext('2d')`) in Canvas drawing.**
3. **Why is native HTML5 Video better than using third-party plugins like Flash?**
4. **What are the privacy implications of the Geolocation API?**
5. **Compare Local Storage to Cookies in terms of capacity and security.**
6. **What is a "Web Worker," and why would a developer use one?**
7. **How does a "Service Worker" enable a website to work without an internet connection?**
8. **Write the JavaScript code to save the string "Score: 100" to Local Storage.**
9. **What is "Microdata," and how does it help with Search Engine Optimization (SEO)?**
10. **Describe the coordinate system of the HTML5 Canvas. Where is point `0,0` located?**

MODULE 10: CAPSTONE APPLICATION

The Social Network: Integrating the Full Stack

Chapter 27: Building "Nexus" – A Professional Social Platform

27.1 Project Architectural Overview

The goal of this capstone is to build Nexus, a lightweight social networking application. This project is designed to demonstrate the "Flow of Data" through all layers of the stack.

The Feature Set:

1. **Identity Layer: Secure Signup and Login with password hashing.**
2. **Persistence Layer: A relational database for users, posts, and friendships.**

3. **Real-Time Layer: AJAX-powered posting (no page refresh).**
4. **Security Layer: Protection against SQLi, XSS, and CSRF.**
5. **Presentation Layer: Mobile-responsive CSS and HTML5 layouts.**

27.2 Phase 1: The Database Schema (The Foundation)

A social network is only as good as its relationships. We need three primary tables.

SQL Schema:

```
CREATE DATABASE nexus_db;

USE nexus_db;


-- Users Table

CREATE TABLE users (

   id INT AUTO_INCREMENT PRIMARY KEY,

   username VARCHAR(50) UNIQUE NOT NULL,

   email VARCHAR(100) UNIQUE NOT NULL,

   password_hash VARCHAR(255) NOT NULL,

   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP

);


-- Posts Table

CREATE TABLE posts (

   id INT AUTO_INCREMENT PRIMARY KEY,

   user_id INT NOT NULL,

   content TEXT NOT NULL,

   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

   FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
```

```
);
```

Architectural Insight: Note the `FOREIGN KEY`. This ensures Referential Integrity. If a user is deleted, their posts are automatically "Cleaned Up" (Cascade Deleted), preventing "Orphaned Data" in our vault.

27.3 Phase 2: The Logic Layer (PHP & Authentication)

We must handle the user's entrance into the system securely.

The Signup Logic (`signup.php` excerpt):

```
<?php
require 'db_config.php';

if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $user = $_POST['username'];
    $pass = $_POST['password'];

    // Technical Requirement: Never store plain-text passwords!
    $hash = password_hash($pass, PASSWORD_DEFAULT);

    $stmt = $conn->prepare("INSERT INTO users (username, password_hash) VALUES (?, ?)");
    $stmt->bind_param("ss", $user, $hash);

    if ($stmt->execute()) {
        header("Location: login.php?msg=success");
    }
```

```
}

?>
```

Logic Breakdown:

1. `password_hash()`: **Uses the Bcrypt algorithm. It adds a "Salt" and hashes the password 10+ times, making it nearly impossible for hackers to reverse-engineer even if they steal the database.**
2. `bind_param()`: **Protects against SQL Injection.**

27.4 Phase 3: The Real-Time Feed (AJAX & JSON)

To make Nexus feel modern, we will use AJAX to submit posts so the user never leaves their feed.

The Frontend (JavaScript):

```
function submitPost() {
    let content = document.getElementById("postContent").value;
    let xhr = new XMLHttpRequest();

    xhr.open("POST", "api_save_post.php", true);
    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

    xhr.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            // Append the new post to the top of the feed
            let feed = document.getElementById("feed");
            feed.innerHTML = "<div class='post'>" + content + "</div>" + feed.innerHTML;
        }
    };
```

```
    xhr.send("content=" + encodeURIComponent(content));
}
```

27.5 Phase 4: Styling and Mobile Responsiveness

Using the CSS Box Model and Flexbox, we ensure the feed looks professional.

```
.post {
    padding: 20px;
    margin-bottom: 15px;
    border-radius: 8px; /* Professional Rounded Corners */
    box-shadow: 0 2px 5px rgba(0,0,0,0.1);
    background: #fff;
}

@media (max-width: 600px) {
    .container { width: 95%; }
}
```

27.6 The Professional Workflow: "Definition of Done"

Before deploying Nexus, a student must perform a "Security Audit":

1. **Input Check: Are all `echo` statements wrapped in `htmlspecialchars()`? (Prevents XSS).**
2. **Session Check: Is `session_start()` at the top of every protected page?**
3. **Logic Check: Can a user delete someone else's post by changing an ID in the URL? (Broken Object Level Authorization).**

27.7 Capstone Final Exercises

1. **Feature Addition: Implement a "Like" button using AJAX and a new `likes` table.**
2. **Security Challenge: Implement a "Lockout" mechanism where a user is blocked**

for 10 minutes after 5 failed login attempts.
3. **UI Challenge:** Use the HTML5 Canvas API to allow users to draw a simple "Profile Sketch" instead of uploading a photo.

27.8 Chapter Review Questions

1. Explain the data flow from the moment a user clicks "Post" until the post appears on their screen in Nexus.
2. Why is `password_hash()` superior to older methods like `md5()` or `sha1()`?
3. Describe how `ON DELETE CASCADE` helps maintain a healthy database.
4. What is the purpose of `encodeURIComponent()` in the AJAX request?
5. How do Sessions maintain the "Logged In" state across different PHP files?
6. Explain the importance of the `user_id` foreign key in the `posts` table.
7. What role does `htmlspecialchars()` play when displaying user-generated posts?
8. How would you modify the CSS to create a "Dark Mode" version of the app?
9. Why should the database connection (`db_config.php`) be in a separate file?
10. Describe the steps to deploy this application from `localhost` to a live web server.